Scrivere prompt che fanno costruire un LLM la vostra applicazione.

BY **MARC GLOOR** · PAGINA 1 DI 2 — NOVE REGOLE ESSENZIALI E IL MODELLO DI RIFERIMENTO

Il successo di un modello nel tradurre un'idea in codice funzionante dipende quasi interamente dalla chiarezza e dalla struttura della richiesta. Studi recenti su requirements engineering e generazione di codice convergono sulla stessa conclusione: prompt di una riga producono software di una riga. Segue un insieme di pratiche — distillato da studi peer-reviewed e guide professionali — per trasformare i requisiti in prompt che un LLM possa effettivamente eseguire.

NOVE REGOLE ESSENZIALI

Dai requisiti al codice, dal terminale al browser — le pratiche che portano l'output dal prototipo alla produzione.

1 Stabilire prima il ruolo e il contesto

Iniziate con un ruolo — «Agisci come un ingegnere backend senior in un'architettura orientata ai servizi» — e un paragrafo che nomini il progetto, i suoi vincoli e gli standard da rispettare. Un ruolo importa implicitamente priorità (sicurezza, scalabilità, stile idiomatico) che altrimenti il modello dovrebbe indovinare.

2 Usare un linguaggio assertivo e specifico

Sostituite «dovrebbe» e «può» con «deve» e «è richiesto». Verbi vaghi generano codice vago. Nominate file, classi, metodi e pattern esistenti che il nuovo codice deve seguire, invece di descriverli in astratto.

> ✗ «Scrivi una funzione per elaborare i dati utente.»
> ✓ «Aggiungi *transformUser* in *UserProcessor.js*, sullo stile di *transformPaymentData*.»

3 Fissare l'ambiente tecnico

Indicate il linguaggio con le versioni esatte, il framework, le librerie ammesse (e vietate), il runtime, la piattaforma di destinazione e le convenzioni di codice. Senza queste informazioni il modello ricade sui pattern di maggioranza dei dati di addestramento, che spesso contraddicono la codebase reale.

4 Strutturare i requisiti come elenco numerato

Scomponete ciò che il software deve fare in voci discrete, numerate e verificabili. Gli elenchi numerati vengono elaborati in modo più affidabile della prosa e vi obbligano ad affrontare le ambiguità prima del modello. Raggruppate separatamente i requisiti funzionali, non funzionali e di interfaccia.

5 Definire input, output e casi limite

Specificate il formato di I/O, pre- e post-condizioni, modalità di errore e almeno due o tre esempi concreti — incluso un caso limite. Gli esempi few-shot superano sistematicamente le specifiche astratte, soprattutto per compiti sensibili alla forma dei dati.

6 Usare il prompting progressivo

Per tutto ciò che va oltre una singola funzione, non chiedete il codice in un colpo solo. Guidate il modello in una catena per fasi: affinare i requisiti, derivare un design, scrivere i test, scrivere il codice. L'output di ogni fase diventa l'input di quella successiva. Questo riproduce il waterfall e offre una superficie di revisione a ogni passo.

7 Integrare la verifica nel prompt

Chiedete al modello di scrivere i test prima — o in parallelo — all'implementazione e di rivedere da solo il risultato rispetto ai vostri criteri di accettazione, elencando le assunzioni e i casi non gestiti. I test sono la forma eseguibile dei vostri requisiti; senza di essi non potete sapere se il codice li soddisfa.

8 Calibrare il rigore al rischio

Uno script usa-e-getta tollera un prompt «pigro»; un flusso di pagamento no. Tarate lo sforzo al peso: i prototipi a basso rischio possono usare un prompting libero ed esplorativo; le funzionalità critiche meritano un passaggio completo spec-poi-codice, con requisiti, design e test persistiti.

9 Specificare il paradigma di interfaccia — TUI o GUI

Nominate esplicitamente l'interfaccia e fissatene le convenzioni; altrimenti il modello sceglierà default che raramente combaciano con la vostra piattaforma.

> **TUI** libreria (textual / curses / prompt_toolkit); schema tasti (vi / emacs / nominato); larghezza minima; --json per le pipe; monocromatico per default.
> **GUI** design system; stati (vuoto, caricamento, errore, successo); accessibilità (WCAG AA); contrasto; parità da tastiera; microcopy nel prompt.

MODELLO DI RIFERIMENTO · UNO SCHELETRO DI PROMPT RIUSABILE

La struttura a otto blocchi che regge nella maggior parte dei progetti.

```
ROLE          Sei un ingegnere {linguaggio} senior. Priorità: chiarezza, testabilità, sicurezza.
CONTEXT       Progetto: {nome, scopo in una riga}. Stile del codice: {pattern, file da rispecchiare}.
STACK         Linguaggio {X.Y}. Framework {X.Y}. Librerie ammesse: {…}. Vietate: {…}.
UI / UX        Paradigma: {TUI | GUI}.
              TUI — libreria {textual/curses/prompt_toolkit}; tasti {vi/emacs/nominati}; larghezza ≥ {N} colonne; --json per le pipe; monocromatico per
default.
              GUI — design system {…}; stati vuoto/caricamento/errore/successo; accessibilità {WCAG AA}; contrasto ≥ 4,5:1; microcopy qui.
REQUIREMENTS  1. {obbligatorio} 2. {obbligatorio} 3. {vietato} … (numerato, assertivo, verificabile).
I/O & EDGES   Input: {tipi, formati}. Output: {tipi, formati}. Casi limite: {…}. Esempi: {…}.
PROCESS       Passo 1 — proponi firma di funzione + docstring; attendi la mia approvazione.
              Passo 2 — scrivi test che falliscono per i requisiti sopra.
              Passo 3 — implementa; poi auto-rivedi rispetto ai requisiti ed elenca le assunzioni.
OUTPUT        Percorsi completi dei file prima di ogni blocco. Pronto per la produzione, senza tronchi, senza segnaposto.
```

EVITARE

Requisiti di una riga. **Confondere il cosa con il come** nella stessa frase. **Lasciare che il modello** inventi lo stack. **Accettare la prima bozza** come definitiva. **Niente test, niente accettazione.**

7×

Più piccole correzioni ma di rilievo emergono quando un prompt è suddiviso in **requisiti → design → test → codice** anziché lanciato in un colpo solo. Il prompting progressivo le intercetta *prima* che raggiungano la codebase.

«Sconsigliamo di affidarsi a metodologie di requisiti su una sola riga per lo sviluppo software basato su LLM. Sottolineiamo invece l'importanza di dedicare tempo e impegno alla formulazione di requisiti dettagliati e di alta qualità.»

ARORA ET AL. · REQUIREMENTS ARE ALL YOU NEED (2024)

FONTI · White et al., *ChatGPT Prompt Patterns for Software Engineering* (arXiv 2303.07839) · Arora et al., *Requirements are All You Need: From Requirements to Code with LLMs* (arXiv 2406.10101) · *Prompt Engineering for Requirements Engineering: A Literature Review & Roadmap* (arXiv 2507.07682) · Ullrich, Koch & Vogelsang, *From Requirements to Code: Developer Practices in LLM-Assisted SE* (arXiv 2507.07548) · *Guidelines to Prompt LLMs for Code Generation* (arXiv 2601.13118) · Mosofsky, *Spec-Then-Code* (GitHub) · Palantir, *Prompt-Engineering Best Practices*.

ESEMPIO · UNA BUILD COMPLETA, DALL'INIZIO ALLA FINE

Costruisci un editor di testo. Copia il modello. Riempi le parentesi.

Sotto due blocchi. **Il primo** *è lo scheletro nudo a otto blocchi di pagina uno — incollatelo nel modello che preferite e sostituite ogni* `{parentesi}` *con le specifiche del vostro progetto.* **Il secondo** *è lo stesso scheletro già compilato per un obiettivo piccolo ma realistico: un editor di testo semplice a singola finestra, basato su browser (una GUI). L'alternativa TUI è descritta nel blocco UI / UX del modello nudo. Usatelo come soluzione di riferimento, oppure copiatelo e adattatelo.*

BLOCK 1 — Il modello nudo — copia questo

SOSTITUISCI OGNI {PARENTESI}

ROLE	Sei un ingegnere {linguaggio} senior. Priorità: chiarezza, testabilità, {priorità}.
CONTEXT	Progetto: {nome + scopo in una riga}. Stile del codice: {pattern, file da rispecchiare}.
STACK	Linguaggio {X.Y}. Framework {X.Y}. Librerie ammesse: {…}. Vietate: {…}.
UI / UX	Paradigma: {TUI | GUI}. TUI — libreria {textual/curses/prompt_toolkit}; tasti {vi/emacs/nominati}; larghezza ≥ {N} colonne; --json per le pipe; monocromatico per default. GUI — design system {…}; stati vuoto/caricamento/errore/successo; accessibilità {WCAG AA}; contrasto ≥ 4.5:1; microcopy qui.
REQUIREMENTS	1. {obbligatorio} 2. {obbligatorio} 3. {vietato} … (numerato, assertivo, verificabile).
I/O & EDGES	Input: {tipi, formati}. Output: {tipi, formati}. Casi limite: {…}. Esempi: {…}.
PROCESS	Passo 1 — proponi struttura dei file + tipi; attendi approvazione. Passo 2 — scrivi test che falliscono per i requisiti. Passo 3 — implementa; auto-revisione; elenca le assunzioni.
OUTPUT	Percorsi completi dei file prima di ogni blocco. Pronto per la produzione, senza tronchi, senza segnaposto.

BLOCK 2 — Esempio svolto — un editor di testo basato su browser

COPIA E ADATTA

ROLE	Sei un ingegnere TypeScript / React senior. Priorità: chiarezza, testabilità, accessibilità. Preferisci codice semplice e idiomatico alle astrazioni furbe.
CONTEXT	Progetto: **NotePad** — un editor di testo a singola finestra che gira nel browser. Stile: componenti piccoli e focalizzati, funzioni pure per le operazioni di testo, niente stato globale. Rispecchia i pattern di `src/components/Toolbar.tsx`.
STACK	TypeScript 5.4. React 18.3. Vite 5. CSS Modules. Librerie ammesse: solo `react` e `react-dom`. Vietate: Redux, MobX, jQuery, qualunque libreria rich-text (Quill, Slate, CKEditor). L'editor DEVE essere un `<textarea>` controllato, non `contenteditable`.
UI / UX	Paradigma: **GUI** (web app a singola finestra). Design: minimale — font di sistema, palette neutra, nessuna decorazione oltre alla toolbar. Stati: documento vuoto → segnaposto centrato «Inizia a scrivere o apri un file»; caricamento → spinner nella toolbar; errore → toast in alto, chiudibile; successo (salvato) → badge «Salvato» per 2 s. Accessibilità (WCAG AA): piena parità da tastiera, nessuna azione solo-mouse; `<textarea>` etichettato; bottoni toolbar con `aria-label`; anello di focus visibile; contrasto ≥ 4,5:1; rispetta `prefers-reduced-motion`. Microcopy: bottone «Salva .txt»; barra di stato «Parole: {n} · Caratteri: {n}»; avviso di chiusura «Ci sono modifiche non salvate. Scartare?»
REQUIREMENTS	1. Mostra il documento in un `<textarea>` a tutta finestra. 2. La toolbar DEVE fornire: Nuovo, Apri (.txt), Salva (scarica .txt), Conteggio parole. 3. Apri DEVE leggere via FileReader e sostituire il documento. 4. Salva DEVE avviare il download del documento come testo UTF-8. 5. Il conteggio parole DEVE aggiornarsi a ogni tasto e comparire in una barra di stato. 6. Il documento DEVE essere persistito in localStorage a ogni modifica e ripristinato al caricamento. 7. Ctrl/Cmd+S DEVE attivare Salva; Ctrl/Cmd+O DEVE aprire il selettore file. 8. L'app DEVE avvisare alla chiusura del tab se ci sono modifiche non salvate (beforeunload).
I/O & EDGES	Input: tasti premuti; file .txt fino a 5 MB. Output: un file .txt scaricato, UTF-8. Casi limite: documento vuoto al primo avvio; ripristino di un file da 4 MB da localStorage; incolla di rifiuti binari (non deve schiantarsi); nomi file con spazi. Esempio: aprire `notes.txt` con «hello\nworld» → la textarea mostra due righe, conteggio parole = 2.
PROCESS	Passo 1 — Proponi la struttura dei file (componenti, hook, utility) e i tipi TypeScript per AppState. Attendi la mia approvazione. Passo 2 — Scrivi test Vitest per: funzione di conteggio parole, derivazione del nome file in salvataggio, round-trip su localStorage, gestione delle scorciatoie da tastiera. Passo 3 — Implementa; poi auto-rivedi rispetto agli otto requisiti sopra ed elenca tutte le assunzioni.
OUTPUT	Percorsi completi dei file prima di ogni blocco (es. `src/App.tsx`). Pronto per la produzione, senza tronchi, senza commenti segnaposto. Per modifiche a file esistenti, restituisci il file completo aggiornato.

Linea rossa = il modello nudo. Incollatelo come punto di partenza e sostituite ogni `{parentesi}`.

Linea verde = un'istanza interamente compilata. Stesso scheletro, reso concreto per un piccolo prodotto.

COME USARLA **1.** Copiate il Blocco 1 nella chat. **2.** Sostituite ogni `{parentesi}` con le specifiche del progetto — il Blocco 2 mostra il livello di dettaglio che funziona. **3.** Inviate il prompt e fermatevi al **Passo 1** per attendere la struttura di file proposta dal modello prima di approvare i Passi 2 e 3. **4.** Trattate il risultato come una bozza: rivedete l'elenco delle assunzioni, eseguite i test, poi iterate.

COMPLEMENTO ALLA PAGINA 1 · Lo scheletro a otto blocchi si appoggia alle stesse fonti citate lì — in particolare Arora et al. (2024) sul prompting progressivo, *Spec-Then-Code* di Mosofsky sulla calibrazione rigore-rischio e le linee guida empiriche di Mastropaolo et al. (2026). L'obiettivo «costruire un editor di testo» è illustrativo; la stessa struttura funziona per uno strumento CLI, un microservizio o una pipeline di dati.